Optimal Hiring of Cloud Servers

A. Stephen McGough[a], Isi Mitrani[b]

[a]*School of Engineering and Computing Sciences, Durham University, DH1 3LE, U.K.*
stephen.mcgough@durham.ac.uk
[b]*School of Computing Science, Newcastle University, NE1 7RU, U.K.*
isi.mitrani@ncl.ac.uk

Abstract

A host uses servers hired from a Cloud in order to offer certain services to paying customers. It must decide dynamically when and how many servers to hire, and when to release them, so as to minimize both the job holding costs and the server costs. Under certain assumptions, the problem can be formulated in terms of a semi-Markov decision process and the optimal hiring policy can be computed. Three situations are considered: (a) jobs are submitted in random batches and servers can be hired for arbitrary periods of time; (b) jobs arrive singly and servers must be hired for fixed periods of time; (c) jobs arrive in bursts of random duration and hiring decisions are made at moments when bursts are initiated or terminated. In all three cases, the optimal policies are compared with some simple and easily implementable heuristics.

Keywords: Cloud Resource Hiring, Semi-Markov decision process, Dynamic resource hiring

1. Introduction

This paper focuses on certain special, and important, dynamic scheduling problems that arise in the market for computer services. It presents a general optimization methodology and applies it in situations where detailed exact and approximate solutions can be developed.

A host offers certain services which involve running user jobs. It does not own servers, but hires them on a temporary basis from a Cloud provider. The host must decide dynamically when, and how many, servers to hire. The objective is to manage optimally the long-term trade-offs between the operating costs (which depend on the number of servers hired), and the

Quality-of-Service, or 'holding' costs (which are proportional to the number of jobs present).

Three distinct models are considered. In the first, jobs are submitted in batches of random size and at random intervals. Servers may be hired and released at arbitrary moments of time, hence the hiring decisions can be taken at the instants when new batches arrive. In the second model, servers must be hired for reasonably long fixed periods of time, e.g. by the hour (this is common among Cloud providers such as Amazon[1]). Hiring decisions are therefore assumed to take place at discrete moments in time, while jobs arrive and depart singly and in continuous time. The third model deals with jobs arriving in bursts of random duration. Several such bursts may be in progress at any one time. The decision moments are the starting points and the finishing points of bursts. The second model is perhaps closer to current practice, but the first and third ones may become more relevant since some Cloud providers are beginning to offer servers for very short-term hire, e.g. by the minute.

We also examine a cost structure which includes a fixed charge per server hired, in addition to the charge proportional to the hire period. Handling such costs requires considerably heavier computational effort.

We show how, under certain assumptions, these dynamic optimization problems can be solved by formulating them in terms of semi-Markov decision processes and applying a policy improvement algorithm. The optimal hiring policy can then be computed in a finite number of iterations. Although that computation is efficient, it may sometimes be too expensive to be carried out on-line. We therefore propose simple and easily implementable heuristic policies whose performance is compared to that of the optimal policy. In fact, in one of the models, a heuristic is shown to be optimal.

The main distinguishing feature of the present study is that we carry out a rigorous dynamic optimization of the systems considered. That is, we consider operational decisions that depend not only on the system parameters, but also on the changing system state. Moreover, the optimization takes into account the transition probabilities between states, and hence covers a long-term system trajectory. This does not appear to have been done before.

Being able to determine the optimal operating policy is valuable, even when good heuristics exist. One may suspect that a simple heuristic policy

[1]https://aws.amazon.com/ec2/pricing/

will perform well, but the only way to quantify such a statement is to compute the optimal policy and carry out a proper comparison.

1.1. Motivation

The models we examine correspond to three different demand scenarios, all of which occur in practice. The first scenario, where jobs arrive in batches, is motivated by applications such as: (i) pattern-matching or searching tasks involving big volumes of data; these can be divided into a number of independent sub-tasks and run in parallel; (ii) simulation studies requiring independent replicated runs in order to derive confidence intervals; (iii) groups of independent workflows prepared by users offline and submitted for execution together.

Examples of existing organisations which have use the Cloud and exhibit batch job submissions include Cycle Computing[2] – a company which acts as a broker offering virtual High-Throughput HTCondor [23] clusters in the Cloud, the high energy physics community [9], and e-Science Central [10], whereby access to Cloud computing resources is offered to users in a transparent manner. As an example Figure 1 shows bursts of jobs which were submitted to e-Science Central as part of the MOVEeCloud project, performing analysis of physical activity. The graph shows the time interval of 12:30pm till 2:30pm on 5th April 2013 with six separate batch submissions. However, at present these systems do not make any attempt to optimize their operating policies.

In the second scenario, jobs arrive singly in a Poisson stream. This occurs when the demand is formed by a superposition of a large number of sources, each of which submits jobs at a low rate. For example, submissions of work to a Cloud photo morphing service made by individuals around the world. The Poisson arrival model is often appropriate, although not always.

The third model, involving bursty arrivals, is motivated by: (i) observations of internet traffic (see, for example, Leland et al., [13]), which have exhibited bursty and heavy-tailed arrival processes; (ii) Cloud bursting where Cloud resources are used to augment an existing internal infrastructure in the cases of extreme demand [11]. The way we have chosen to model burstiness is by a special Markov-modulated Poisson process (MMPP). Such models have been used before (e.g., see Rajabi and Wong [21] and Revzina [22]), but

[2]http://www.cyclecomputing.com

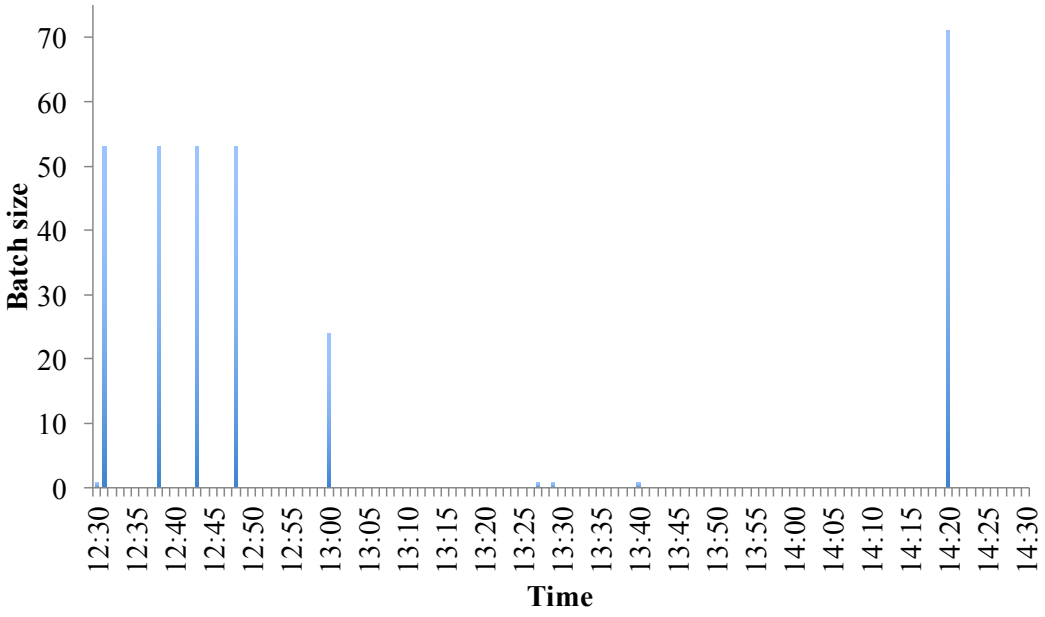

Figure 1: Batch workload submission in e-Science Central

not for the purpose of optimizing the number of servers.

1.2. Related Work

The existing approaches to the server hiring problem are, on the whole, concerned with static policies. That is, the hiring decisions are based on knowing or estimating the characteristics of user demand. Those decisions change only when the demand parameters change. On the other hand, a dynamic policy reacts to random changes in the system state, even if the demand characteristics remain the same. In general, dynamic policies are more efficient than static ones, as we shall see when presenting our numerical results.

In previous work McGough et al. [17] evaluate a number of simple heuristics for the server hiring problem, evaluating these using trace-logs. Mazzucco et al. [14] have used workload estimation in order to determine the optimal number of servers to hire. By assuming that impatient users will abandon job requests (common for HTTP) an Erlang-C problem is converted into an Erlang-A problem and a solution is obtained by a binary search algorithm. This work is extended in [16] to evaluate the number of Virtual Machines (VMs) required by a Software-as-a-Service (SaaS) provider using

an Infrastructure-as-a-Service (IaaS) backend. Bodík et al. [2] use statistical machine learning to estimate the workload in the next epoch. Like other approaches, this requires additional servers to be provisioned in case the estimate is low.

Another static version of the server hiring problem was considered by Lampe et al. [12], who examined the optimal placement of a fixed set of jobs, with given run times and resource requirements, onto different Cloud servers. An exact formulation based on Binary Integer Programming and an approximate algorithm using bin-packing techniques were proposed. A similar problem involving workflows was addressed by Byun et al. [4, 5]. In this instance, the servers are not different, but the jobs must satisfy a set of precedence constraints. Again, the aim is to minimize the cost of executing a given workflow on the Cloud. An approximate scheduling algorithm is proposed.

Chaisiri et al. [7] attempt to exploit the lower costs of future reservations in order to minimize the overall cost of hiring Cloud resources. They use stochastic and deterministic programming techniques, coupled with sample-average approximations or Benders decomposition. This study has some dynamic aspects. However, the actual demand process is not modelled and therefore the costs of waiting cannot be taken into account.

Wang et al. [26] studied the resizing of service centers in response to changes in workload, taking into account different degrees of traffic burstiness by means of bounds and approximations. Their optimization criterion does not include holding costs and the evolution of the queueing process is not considered.

The server hiring problem is also related to other server allocation topics, for which a large body of literature exists. These topics include the trade-offs between performance and power consumption in a service center. In Mazzucco et al. [15] and Mitrani [18], certain dynamic server allocation policies were analysed, but no attempt was made to find the optimal policy. The maximization of throughput and the minimization of waiting or response time were considered in Urgaonkar et al. [25], Chandra et al. [6] and Bennani and Menascé [1].

The general Semi-Markov decision process and the algorithm for computing the optimal policy are described in section 2. The applications of the theory to the models with batch arrivals, fixed hiring periods and bursty arrivals are presented in sections 3, 4 and 5, respectively. Section 6 considers the effect of introducing a fixed component into the cost of hiring servers.

Section 7 contains the results of several numerical experiments where the optimal policy is evaluated and compared with some heuristics. Several directions for further research are outlined in the conclusion.

2. Semi-Markov decision processes

Consider a finite-state system which is observed at random points in time, t_i $(i = 0, 1, \ldots)$. These instants are called 'decision epochs' and the intervals between them are 'decision intervals'. If at time t_i the system is in state j $(j = 1, 2, \ldots, J)$, an action, or decision, a_j, is taken. That action may influence the length of the ensuing decision interval, $t_{i+1} - t_i$, and also the system state at the next epoch. However, neither the decision interval nor the next state depend on anything that happened prior to t_i. Such a process is called a 'semi-Markov decision process'. The actions taken in the various states constitute a 'stationary policy', if for all states j, whenever the state j is observed, the same action, a_j, is taken, regardless of current time and past history.

The system incurs costs which depend on the states it passes through and on the decisions taken in those states. Let $Z_A(t)$ be the total cost incurred up to time t under a stationary policy A. Then the long-run average cost of policy A per unit time is defined as the limit:

$$g(A) = \lim_{t\to\infty} \frac{1}{t} E[Z_A(t)] \; . \tag{1}$$

That quantity, which does not depend on the initial state when the embedded Markov chain is irreducible and aperiodic, is the optimization criterion. The object is to find a policy A that minimizes $g(A)$.

The evolution of the process under the control of a stationary policy A is governed by the succession of states at decision epochs, the decisions made at those epochs and the costs incurred during the decision intervals. Let $p_{j,k}(A)$ be the transition probability that the system will be in state k at the next decision epoch, given that the current state is j and the policy is A; $j, k = 1, 2, \ldots, J$. Also, denote by $c(j, A)$ the average cost incurred during a decision interval, given the current state j and policy A. Finally, let $\tau_j(A)$ be the average length of the decision interval, given the current state and policy.

The long-run average cost of policy A, $g(A)$, can be computed by introducing certain quantities called 'relative values', v_j, $j = 1, 2, \ldots, J$, (see

Tijms [24]). These relative values, together with $g(A)$, satisfy a set of simultaneous linear equations:

$$v_j = c(j, A) - \tau_j(A)g(A) + \sum_{k=1}^{J} p_{j,k}(A)v_k \ \ ; \ \ j = 1, 2, \ldots, J \ , \qquad (2)$$

with $c(j, A)$, $p_{j,k}(A)$ and $\tau_j(A)$ as defined above.

In this set, there are J equations with $J + 1$ unknowns. However, if the same constant, c, is added to all relative values v_j, the value of $g(A)$ would not change (since for each j, the sum of $p_{j,k}(A)$ with respect to k is 1). Therefore, the solution of (2) can be made unique by choosing an arbitrary state, m, and setting $v_m = 0$.

The optimal policy can be determined by the following 'policy improvement' algorithm.

1. Choose some stationary policy A.
2. Compute $g(A)$ and v_j by solving (2).
3. For each j, find the action a^* that minimizes the right-hand side of equation (2):

$$\min_a \left[c(j, A) - \tau_j(a)g(A) + \sum_{k=1}^{J} p_{j,k}(a)v_k \right] \ ,$$

   where $g(A)$ and v_k keep the values already computed.
4. If the new actions a^* are the same as the old ones for all states, i.e. if the new policy A^* is the same as A, stop. Otherwise repeat from step 2, replacing A with A^*.

This algorithm terminates after a finite number of iterations, producing an optimal policy and the corresponding long-run average cost, g.

An efficient and stable method for solving the set of equations (2) is to use Gauss-Seidel iterations, starting with $v_j = 0$ for all j. We have always found these iterations to converge. Intuitively, this is because the sum in the right-hand side of (2) is a convex combination of the v_k values from the previous iteration and so the new values stay bounded. If that method is adopted, then the complexity of computing the optimal policy is on the order of $O(J^2SI)$, where J is the size of the state space, S is the number of iterations in the Gauss-Seidel solution and I is the number of iterations in the policy-improvement algorithm.

3. Batch arrivals

The first system we examine is one where user demands arrive at the host's site in a Poisson stream with rate λ. Consecutive demands consist of batches of jobs whose sizes are i.i.d. random variables with an arbitrary distribution. Let b_j be the probability that a batch contains j jobs ($j = 1, 2, \ldots, ...$). The average batch size is denoted by b:

$$b = \sum_{j=1}^{\infty} j b_j \ . \tag{3}$$

A job's runtime, on any available server, is distributed exponentially with mean $1/\mu$. Thus, the total offered load at the site is $\rho = \lambda b/\mu$. When all available servers are busy, jobs wait in a common FIFO queue. Servers may be hired and released at any moment.

In this model, the decision epochs are the instants just after the arrival of a new batch. The system state at a decision epoch is the total number, j, of jobs present. That number may include jobs from previous batches that are still waiting or are in service. The decision taken at a decision epoch is the number of servers, n, that are hired from a Cloud provider and will be available to serve jobs. That number may include previously hired servers, plus any newly hired ones, or minus any servers whose hire is terminated at this decision epoch.

Each job present incurs a holding cost of c_1 per unit time spent in the system. These costs reflect the importance attached to fast service. In addition, each hired server incurs a cost of c_2 per unit time. This is predicated on the assumption that the host is dealing with a Cloud that allows hire and release at arbitrary moments, with charges proportional to the duration of hire. A different hire regime will be modeled in the next section.

Thus, the total cost incurred per unit of time during which there are j jobs present and n servers hired is $c_1 j + c_2 n$.

Note that in this model the decision interval does not depend on the current state or on the decision taken. The average length of that interval is the average interarrival time between batches: $\tau = 1/\lambda$.

Since the algorithms available for determining the optimal policy require that the state space is finite, we assume that there is an upper bound, J, for the number of jobs that may be present. If an incoming batch would cause that bound to be exceeded, some or all of its jobs are rejected. That

condition is not too restrictive: under any policy that does not allow the queue to saturate, one can choose J sufficiently large so that the probability of rejecting jobs is negligible. However, the numerical complexity of the solution increases with J.

To write equations (2) for a given policy A in the present model, we need expressions for $c_j(n)$ and $p_{j,k}(n)$, where n is the number of servers hired in state j under policy A. We start with the costs. Let $T_j(n)$ be the total average time that the j jobs currently present spend in the system during the decision period, given that n servers are available to serve them. There are two cases to consider:

1. If $j \leq n$, all jobs present are being served. The contribution of each job to $T_j(n)$ is the average minimum of its remaining service time and the remaining decision period. Hence, in this case,

$$T_j(n) = \frac{j}{\lambda + \mu} \;\; ; \;\; j = 1, 2, \ldots, n \; . \tag{4}$$

2. If $j > n$, then n jobs are being served and $j - n$ are waiting. The next event to occur is either a service completion, with probability $n\mu/(\lambda + n\mu)$, or an arrival of a new batch, with probability $\lambda/(\lambda + n\mu)$. The average interval until that event is $1/(\lambda + n\mu)$, and there are j jobs present during it. If the next event is a service completion, then the decision period continues with $j-1$ jobs present; otherwise it terminates and there is no further contribution to $T_j(n)$. This provides a recurrence relation,

$$T_j(n) = \frac{j}{\lambda + n\mu} + \frac{n\mu}{\lambda + n\mu} T_{j-1}(n) \;\; ; \;\; j = n+1, n+2, \ldots, J \; . \tag{5}$$

Equation (4), together with the recurrences (5), allow the holding times $T_j(n)$ to be computed easily for all j and n. The average cost, $c(j, n)$, incurred during a decision period is the sum of the holding cost and the server cost:

$$c(j, n) = c_1 T_j(n) + c_2 n \frac{1}{\lambda} \; . \tag{6}$$

Before addressing the transition probabilities $p_{j,k}(n)$, consider the probability, $q_{j,k}(n)$, that there will be k jobs present *just before* the next decision epoch, given that there are j jobs now and n servers are available. That is the probability that $j - k$ jobs are completed during the decision interval. There are three distinct cases:

1. If $j < n$, more servers become idle with each departing job. In order that k jobs are left at the end of the decision period, the latter must terminate when there are k busy servers. Hence,

$$q_{j,k}(n) = \left[\prod_{i=k+1}^{j} \frac{i\mu}{\lambda + i\mu}\right] \frac{\lambda}{\lambda + k\mu} \;\; ; \;\; k = 0, 1, \ldots, j \; , \tag{7}$$

   where an empty product is equal to 1 by definition.
2. If $j \geq n$ and $k \geq n$, then $q_{j,k}(n)$ is the probability that exactly $j-k$ jobs are completed by n busy servers before the decision period terminates:

$$q_{j,k}(n) = \left[\frac{n\mu}{\lambda + n\mu}\right]^{j-k} \frac{\lambda}{\lambda + n\mu} \;\; ; \;\; k = n, n+1, \ldots, j \; . \tag{8}$$

3. If $j \geq n$ and $k < n$, then of the $j - k$ completions that must take place before the end of the observation period, $j - n + 1$ occur while n servers are busy and $n - 1 - k$ with gradually diminishing number of busy servers:

$$q_{j,k}(n) = \left[\frac{n\mu}{\lambda + n\mu}\right]^{j-n+1} \left[\prod_{i=k+1}^{n-1} \frac{i\mu}{\lambda + i\mu}\right] \frac{\lambda}{\lambda + k\mu} \;\; ;$$

$$k = 0, 1, \ldots, n-1 \; . \tag{9}$$

Now we can obtain the transition probabilities from state j to state k, $p_{j,k}(n)$, by remarking that the number of jobs present after the arrival of the next batch is the convolution of the number left over at the end of the decision interval and the number contained in the new batch. Hence,

$$p_{j,k}(n) = \sum_{i=0}^{m} q_{j,i}(n) b_{k-i} \;\; ; \;\; k = 1, 2, \ldots, J-1 \; , \tag{10}$$

where $m = \min(j, k-1)$. The exception to that pattern is destination state J, which may be reached after rejecting some new arrivals:

$$p_{j,J}(n) = \sum_{i=0}^{j} q_{j,i}(n) \sum_{s=J-i}^{\infty} b_s \; . \tag{11}$$

All quantities necessary for setting up equations (2), and hence for applying the policy improvement algorithm, are now available.

Since servers can be hired and released at any point, one could reduce costs slightly by releasing servers as soon as they are no longer needed, even if the current decision period has not expired. This improvement is easy to implement, but its effect is rather marginal. When good hiring decisions are made, servers are rarely idle.

4. Fixed hiring periods

We now address a system where a server must be hired for a sizeable minimum period of time, τ. Amazon, for example, hires servers by the hour[3]. Although in principle one could initiate a hire at any time, it is reasonable, and more tractable, to use the instants 0, τ, 2τ, ..., as decision epochs (i.e., the length of the decision interval is τ). Assume that jobs arrive singly during a decision interval, in a Poisson stream with rate λ. Their service times are again distributed exponentially, with mean $1/\mu$.

Thus, if there are j jobs in the system at a decision epoch, and n servers are hired, then during an interval of length τ the queue behaves as a transient $M/M/n/J$ queue (J is the bound on the number of jobs present), with initial state j. To define our decision process, we need the transition probabilities, $p_{j,k}(n)$, that there will be k jobs at time τ, given that there were j jobs at time 0 and n servers were hired.

Denote by $P(t) = [p_{j,k}(t)]$, $j, k = 0, 1, \ldots, J$, the transient transition probability matrix for the $M/M/n/J$ queue over the interval (0,t). Clearly, $P(0) = I$, where I is the $(J+1) \times (J+1)$ identity matrix. We are interested in computing the j'th row of $P(\tau)$.

Let G be the generator matrix for the $M/M/n/J$ queue:

$$G = \begin{bmatrix} -\lambda & \lambda & & & \\ \mu_1 & -(\lambda+\mu_1) & \lambda & & \\ & & \ddots & & \\ & & & -(\lambda+\mu_{J-1}) & \lambda \\ & & & \mu_J & -\mu_J \end{bmatrix}, \tag{12}$$

where $\mu_i = \min(i, n)\mu$. The matrix $P(t)$ is given by the matrix-exponential:

$$P(t) = e^{Gt}\ . \tag{13}$$

[3]https://aws.amazon.com/ec2/pricing/

If the solution algorithms are implemented in Matlab, this matrix exponentiation can be performed by the built-in function $expm(G * t)$, which is stable and fast. If that is not available, one could employ the 'uniformization' technique, which involves replacing the continuous-time Markov process with an equivalent discrete-time Markov chain using the parameter $\gamma = \lambda + n\mu$ (e.g., see [20]). The generator matrix G is replaced by the matrix:

$$Q = I + \frac{G}{\gamma} ,$$

where I is the identity matrix. Then $P(t)$ is given by the series:

$$P(t) = \sum_{i=0}^{\infty} Q^i \frac{(\gamma t)^i}{i!} e^{-\gamma t} . \tag{14}$$

This expression provides an efficient way of computing $P(t)$ because (a) Q is a stochastic matrix, so the elements of Q^i remain uniformly bounded for all i (since the rows always sum up to 1), and (b) the Poisson probabilities that appear in (14) converge rapidly to 0. Hence, the infinite series can be truncated on the right, and possibly on the left, resulting in a finite sum:

$$P(t) = \sum_{i=\ell}^{r} Q^i \frac{(\gamma t)^i}{i!} e^{-\gamma t} , \tag{15}$$

where ℓ and r are chosen so that the two omitted tails are negligible (see [8]).

It remains to determine the average cost, $c_j(n)$, incurred during a decision interval. Let L_j be the average number of jobs in the system at time τ, given that there were j jobs at time 0 and n servers were hired. That average is obtained from:

$$L_j = \sum_{k=1}^{J} k p_{j,k}(\tau) . \tag{16}$$

The average number of jobs present *during* the decision interval can be approximated by taking the mean of the queue sizes at the beginning and end of the interval, i.e. $(j + L_j)/2$. Hence, the total cost incurred during the interval is given by:

$$c(j, n) = \left[c_1 \frac{j + L_j}{2} + c_2 n \right] \tau . \tag{17}$$

Using these expressions, the optimal policy can be computed as described in section 2.

5. Bursty arrivals

Our third model is concerned with a system where the demand consists of bursts of jobs, whose starting points form a Poisson stream with rate γ. The burst durations are i.i.d random variables distributed exponentially with mean $1/\nu$. Several bursts can be in progress simultaneously. The jobs associated with each burst arrive in a Poisson stream with rate λ; their run times (on any server) are i.i.d random variables distributed exponentially with mean $1/\mu$. Hence, if there are j bursts in progress, the total job arrival rate is $j\lambda$ and the total offered load is $j\rho = j\lambda/\mu$. When all available servers are busy, jobs wait in a common FIFO queue.

Hiring decisions are made when new bursts start or old ones terminate. The system state at a decision epoch, i.e. just after the start or the termination of a burst, is the number of bursts currently in progress, j. The decision taken at that epoch is the number of servers, n, that are hired and will be available to serve jobs. As before, that number may include previously hired servers, plus newly hired ones or minus those whose hire is terminated.

It is assumed that γ and ν are small, compared to λ and μ. In other words, the intervals between decision epochs are assumed to be long, compared to job interarrival and service times. This is not unreasonable. Consequently, if the system is in state j and decision n is taken, then the queue can be treated as an $M/M/n$ queue in the steady state, with arrival rate $j\lambda$ and service rate μ. That queue would be stable if $j\lambda < n\mu$.

In order that the state space is finite, a limit J is imposed on the number of bursts that can be in progress at any one time. New bursts that attempt to start when the system is in state J are rejected. Alternatively, a limit N may be imposed on the number of servers that may be hired. In that case, J would be determined by the stability requirement $J\lambda < N\mu$.

Note that this arrival model is a special case of a Markov-modulated Poisson process (MMPP). Such processes have been used already to model bursty arrivals, e.g., see [21, 22]. We could have used a general MMPP model, but have chosen this version because of its intuitive interpretation.

If the system is in state j at a decision epoch, then the interval until the next decision epoch is distributed exponentially, with mean $\tau_j = 1/(\gamma + j\nu)$. At the next epoch, the state will be $j + 1$ with probability $\gamma\tau_j$, except if $j = J$, in which case it will remain in state J with that probability. The next state will be $j - 1$ with probability $j\nu\tau_j$, except if $j = 0$, in which case that transition cannot occur. The average cost, $c(j, n)$, incurred *per unit time*

during a decision interval in which the system is in state j and a decision n is taken, is given by:

$$c(j,n) = c_1 L_j + c_2 n \ , \tag{18}$$

where L_j is the steady-state average number of jobs in the $M/M/n$ queue with arrival rate $j\lambda$ and service rate μ. The average cost incurred over the decision interval is $c(j,n)\tau_j$.

Although the policy improvement algorithm can now be applied, a short cut exists to the determination of the optimal hiring policy in this model. It is made possible by a particular feature, which simplifies the optimization considerably. It is that the state transition probabilities $j \to j+1$ and $j \to j-1$ do not depend on the decisions taken. Hence, the steady-state probabilities, π_j, that the system is in state j, are independent of the hiring policy. The long-run average cost, g, of a given policy can be written as:

$$g = \sum_{j=0}^{J} \pi_j c(j,n) \ . \tag{19}$$

This equation is valid for any Markov decision model under a stationary policy. However, it is not often useful because in general the distribution π_j depends on the policy and cannot easily be determined. In the present case, the bursts in progress behave like the calls in an Erlang loss system (see [18]) with parameters γ, ν and J. Hence, π_j are the steady-state Erlang probabilities:

$$\pi_j = \frac{\sigma^j}{j!} \left[\sum_{k=0}^{J} \frac{\sigma^k}{k!} \right]^{-1} \quad ; \quad j = 0, 1, \ldots, J \ , \tag{20}$$

where $\sigma = \gamma/\nu$.

Since the probabilities π_j do not depend on the policy, equation (19) implies that the policy which, for each j, chooses n to minimize the one-step cost $c(j,n)$, is optimal. Indeed, the right-hand side of (19) shows that the value of g achieved by this 'greedy' policy cannot be improved.

In our other models, where the state transitions depend on the decisions taken, the above argument cannot be applied.

The assumptions in this section can be generalized in two directions. First, job arrivals could be governed by an arbitrary MMPP, provided that the modulating environment changes state rarely, compared to the local arrival rates. Since hiring decisions made at the points when the environment

changes state do not affect its evolution, the optimality of the greedy policy would continue to hold. Second, the burst durations need not be distributed exponentially. The Erlang model is 'insensitive', so (19), (20) would remain valid if that distribution was general.

6. Fixed cost component

In each of the models considered here, in addition to the running server costs (c_2 per server per unit time), one may wish to introduce unit costs for hiring servers. Suppose that, at the moment a server is hired, a cost of c_3 is incurred. Thus, if a server is hired for a period of time t, its total cost would be $c_3 + c_2 t$. The fixed component may reflect some set-up costs, or it may be imposed by the provider in order to discourage hiring for very short periods.

This modification changes substantially the optimization procedure. In order to keep track of the number of new hirings, the system state must be described by a couple (j, s), where j is the number of jobs or bursts present and s is the number of servers currently hired. If that state is observed and decision n is taken, then the number of newly hired servers is $n - s$ when $n > s$ and 0 when $n \leq s$. Consequently, the average cost incurred during the ensuing decision period would be:

$$c(j, s, n) = c_3 \max(n - s, 0) + (c_1 L + c_2 n)\tau \ , \tag{21}$$

where L is the average number of jobs present in the particular model and τ is the average length of the decision period. Moreover, if the number of jobs (or bursts) experiences a transition from j to k, then at the next decision instant the system state will be (k, n). Hence, the evaluation of the transition probabilities does not require any new analysis.

The state space is now much larger, but one can still write equations analogous to (2) and apply the policy improvement algorithm in order to determine the optimal policy. The only difference will be an increased computational complexity.

Note that the result of section 5, namely that in the case of the model with bursty arrivals the greedy policy is optimal, ceases to hold with the new cost structure. This is because the next state, (k, n), now depends on the decision taken, n, and therefore the steady-state distribution of the system state depends on the operating policy.

7. Heuristics and experiments

When the computation of the optimal policy becomes expensive, it may be worth exploring policies that are sub-optimal, but offering good performance and ease of implementation.

A promising heuristic policy for any given model is the one which, at every decision epoch, minimizes the average cost incurred during the current decision interval. In other words, when the current state is j, take the action n^* such that:

$$c(j, n^*) = \min_n c(j, n) \ , \tag{22}$$

where $c(j, n)$ is the cost appropriate to the model. This will be called the 'greedy' heuristic, as it looks only at the current decision interval and does not care about the future.

The implementation of the greedy heuristic does not require any iterations; it is enough to evaluate the costs $c(j, n)$ for different values of n. In practice, that heuristic is orders of magnitude faster to determine than the optimal policy.

We saw in section 5 that the greedy heuristic is optimal for the model with bursty arrivals. In the other models it is not optimal, but it will be interesting to see how close it gets.

In addition, an even simpler policy will be introduced to use as a benchmark. The latter abandons dynamic decision-making altogether and hires a fixed number of servers, n^*, regardless of the system state. This is, in fact, the policy normally adopted in practice. To avoid saturating the queue, n^* should be chosen so that the average long-term server occupancy is less than 100%. For example, one could aim for an occupancy of 70%. In the case of batch arrivals, bearing in mind that the offered load is $\rho = \lambda b/\mu$, where b is the average batch size, the above condition implies:

$$n^* = \left\lceil \frac{\lambda b}{0.7\mu} \right\rceil \ . \tag{23}$$

For the second model, the offered load is $\rho = \lambda/\mu$, so the allocation becomes:

$$n^* = \left\lceil \frac{\lambda}{0.7\mu} \right\rceil \ . \tag{24}$$

That policy will be referred to as the 'fixed policy'.

Figure 1 illustrates and compares the behaviour of the three policies for the batch arrivals model, in the case where batch sizes are distributed geometrically with parameter α. That is, the probability that a batch contains j jobs is $\alpha(1-\alpha)^{j-1}$. The average batch size is $b = 1/\alpha$. The offered load is increased by decreasing α, and the long-term average cost, g, is plotted against the average batch size. The average service time is $1/\mu = 1$, while the batch arrival rate is $\lambda = 0.1$. In this experiment, it was assumed that the unit holding cost and the unit server cost are equal: $c_1 = c_2 = 1$.

The bound on the number of jobs in the system was taken as $J = 100$. Under all three policies, the probability of reaching that bound is small. For example, when the average batch size is 50, the probability that a batch of size 100 will be submitted is about 0.1.

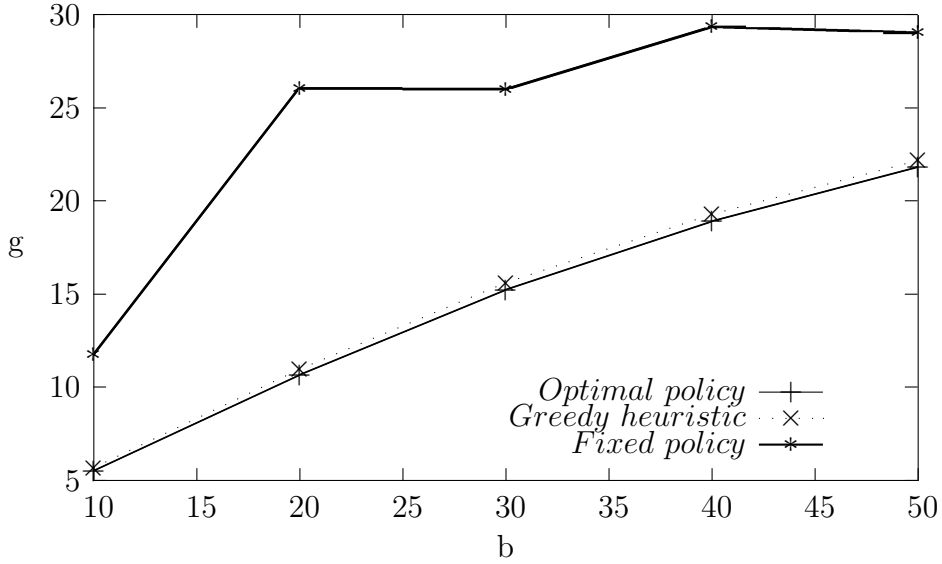

Figure 2: Batch arrivals: geometric batches
Average cost against average batch size

A notable feature of the figure is that the greedy heuristic is almost optimal over the entire range of offered loads. One would therefore be justified in using the heuristic in practice, knowing that its performance cannot be improved significantly. By contrast, the costs of the fixed policy are considerably higher. That remains the case if the 70% occupancy of the

servers is replaced by 80% occupancy. Of course, the more the fixed policy over-provides servers unnecessarily, the poorer its performance would be. The non-monotone character of the graph for the fixed policy is due to the rounding-up operation in (23).

It is interesting to compare the run time of the policy improvement algorithm with the one computing the greedy heuristic. This is done in the table below, for different values of the threshold J.

Table 1: Run times (seconds)

J	20	40	60	80	100	120
Greedy	0.0	0.1	0.4	1.0	2.2	3.3
Optimal	0.9	11.8	85.8	180.1	640.8	1327.0

The table shows that computing the optimal policy tends to be more than two orders of magnitude slower than finding the greedy allocation. The size of the state space, determined by the threshold J, is a major factor in the execution times.

Intuitively, the higher the dependency between system states at consecutive decision instants, the more important it is to look into the future when making decisions. Hence, when that dependency increases, one can expect the greedy heuristic to do less well.

This tendency is confirmed by figure 2. Here, the optimal policy and the greedy heuristic are compared for increasing values of λ, i.e. for decreasing intervals between decision instants. The average batch size is kept at $b = 10$, and the other parameters are as in figure 1.

When $\lambda = 0.1$, the average decision interval is ten times as large as the average service time. At the other end of the range, it is only twice as large, implying stronger dependency between consecutive states and making the short-sighted greedy heuristic less efficient. However, even in this case (somewhat unlikely to occur in practice), the difference between the heuristic and the optimal policy is less than 20%.

Next, we experiment with a batch size distribution that has been constructed to have a large coefficient of variation. More precisely, batches consist of a single job with probability 0.7, and B jobs with probability 0.3. The average batch size is $b = 0.7 + 0.3B$. The coefficient of variation grows roughly linearly with B. In figure 3, B is varied between 20 and 100, and the average achieved cost is plotted against b.

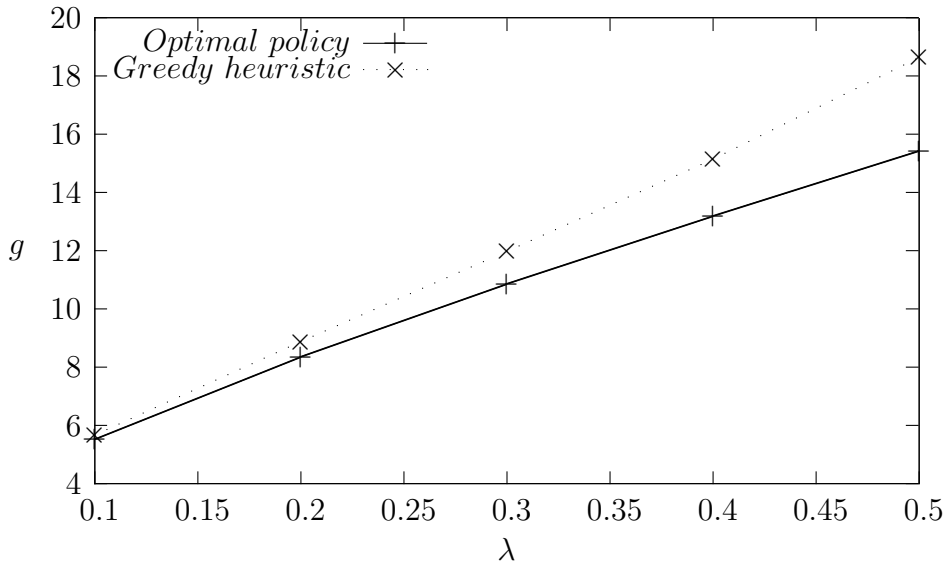

Figure 3: Batch arrivals: Average cost against arrival rate

It seems that large coefficients of variation do not prevent the greedy heuristic from performing well. Its costs are almost indistinguishable from those of the optimal policy. On the other hand, the fixed policy is, if anything, worse than before in comparison.

In the fourth experiment, the characteristics of the demand are held fixed, at $\lambda = 0.1$, $\mu = 1$, $\alpha = 0.04$ (i.e., average batch size of 25). Also, the unit server cost is fixed at $c_2 = 10$. What is varied is the unit holding cost, from $c_1 = 5$ to $c_1 = 20$. That is, the relative cost of keeping jobs in the system is varied from half to double the cost of a server.

The results are shown in figure 4, where the average long-term costs g achieved by the optimal policy, the greedy heuristic and the fixed policy are plotted against c_1.

Again, it is notable that the greedy heuristic achieves nearly optimal costs over the entire range of c_1 values. By contrast, the performance of the fixed policy is rather poor. Moreover, whereas the cost of the fixed policy grows linearly with c_1 (as can be expected), those of the optimal and greedy policies grow slower than linearly.

It is perhaps worth pointing out that, for all points in these three figures, the policy improvement algorithm took no more than 3 iterations to find the

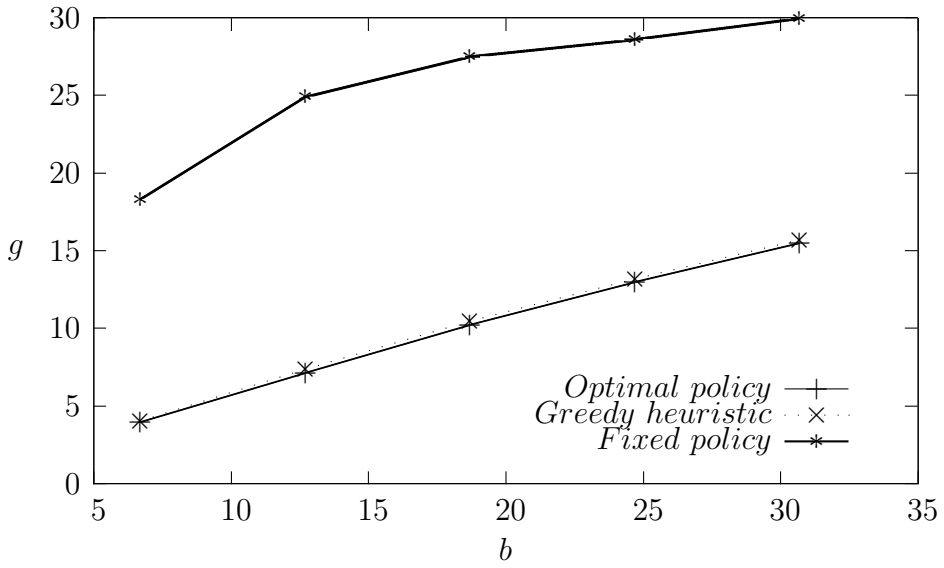

Figure 4: Batch arrivals: skewed batch distribution
Average cost against average batch size

optimal policy.

The next three experiments concern the model with fixed hire periods. In figure 5, the offered load is increased from $\rho = 10$ to $\rho = 18$ by varying the job arrival rate. The service rate is kept at $\mu = 1$, and the unit holding cost is half of the server cost: $c_1 = 0.5$, $c_2 = 1$. The bound on the number of jobs is $J = 50$. The hire period length is $\tau = 4$, which means that on the average, between 40 and 72 jobs arrive during a decision period. The fixed policy is based on equation (24).

We observe that the difference between the worst policy (fixed) and the best one (optimal) is now much narrower. This is due to the fact that jobs continue to arrive throughout a decision period, and the rate of arrivals does not depend on the action taken. This reduces the advantages derived from making dynamic decisions. The costs achieved by the optimal policy are about 15% lower than those of the fixed policy. The greedy heuristic still performs quite well, but its costs are now about 10% higher than those of the optimal policy.

In figure 6, the job arrival rate is kept fixed at $\lambda = 12$. The service rate, server cost and decision period length have the same values as before, $\mu = 1$,

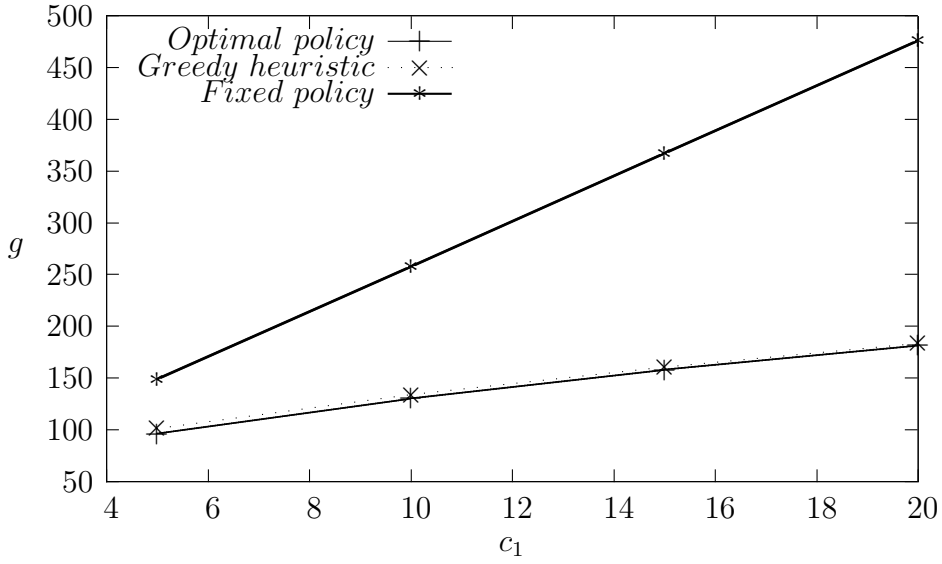

Figure 5: Batch arrivals: Average cost against unit holding cost

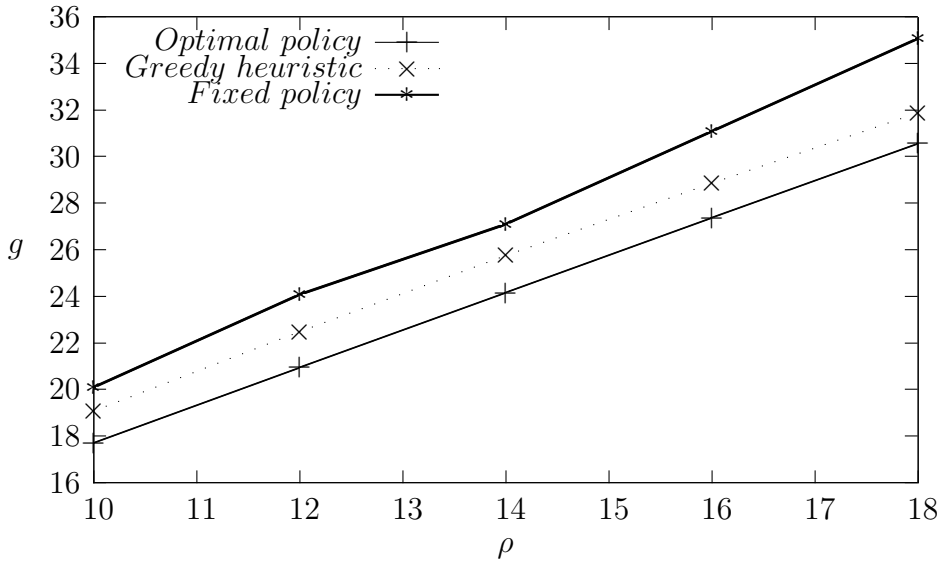

Figure 6: Fixed hiring periods: Average cost against offered load

$c_2 = 1$, $\tau = 4$, while the unit holding cost is varied from half to twice the server cost: $0.5 \leq c_1 \leq 2$.

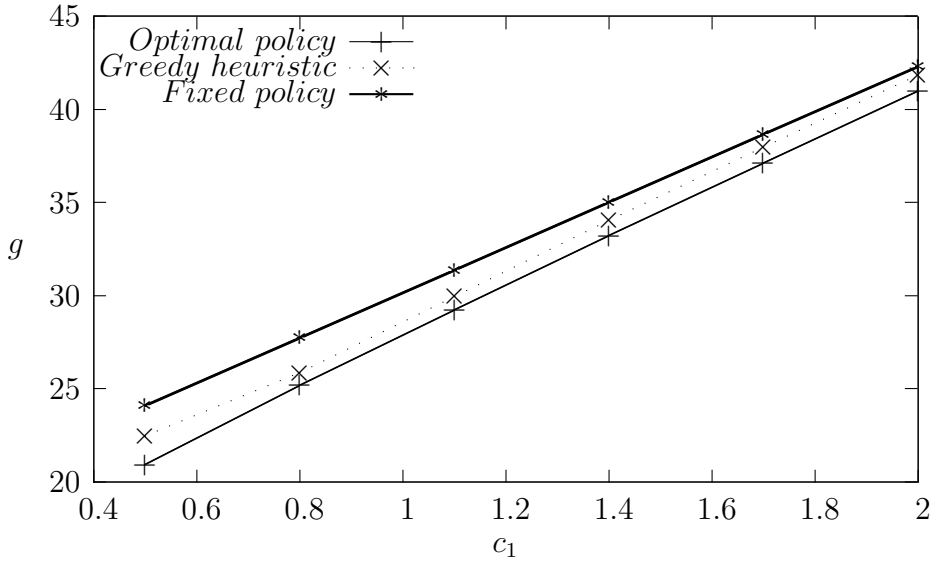

Figure 7: Fixed hiring periods: Average cost against unit holding cost

The average costs achieved by the three policies are quite close over the entire range of c_1 values. Moreover, it is notable that the higher the value of c_1 relative to c_2, the closer those costs are, i.e. the lower the benefit of dynamic decision-making. Indeed, one could have expected that when the dominant factor is customer performance, the most important part of the policy is to always maintain enough servers to cope with the load.

In the next experiment, traffic characteristics and unit costs are kept fixed ($\lambda = 12$, $\mu = 1$, $c_1 = c_2 = 1$), while the length of the decision interval is varied from $\tau = 2$ to $\tau = 10$. That is, the average number of arrivals during a decision interval varies from 24 to 120.

The fixed policy is independent of τ, so its graph is a horizontal line. The optimal and greedy policies also approach a horizontal asymptote. This is predictable, since the system tends to reach steady state during a large decision interval, and the distribution at the next decision epoch becomes independent of the current state. For the same reason, the greedy heuristic, whose performance can be worse than that of the fixed policy for very short

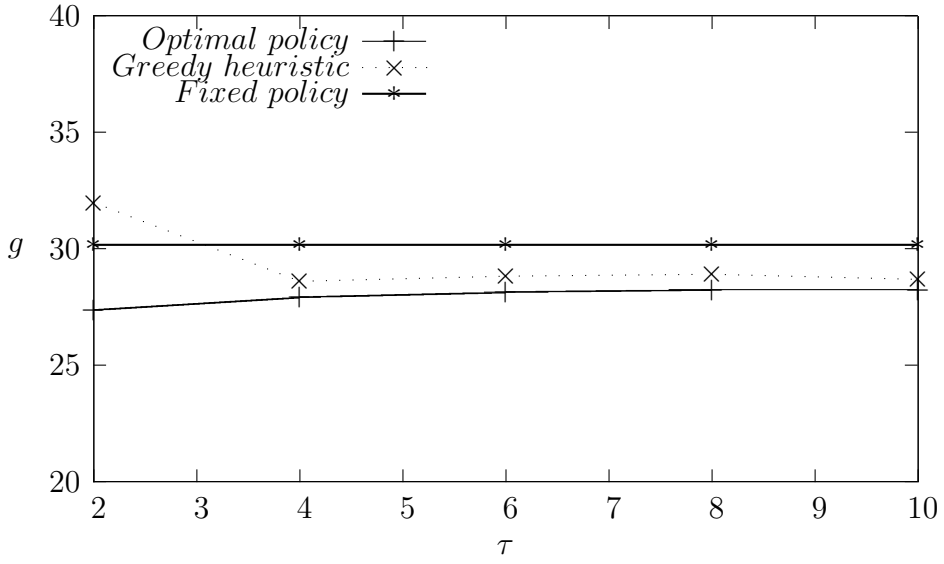

Figure 8: Fixed hiring periods: Average cost against hire period length

decision intervals, becomes not only 'nearly optimal', but optimal, in the limit $\tau \to \infty$.

For all points in the last three figures, the policy improvement algorithm again took no more that 3 iterations to find the optimal policy.

The final figure concerns the model with bursty arrivals. Here we know that the greedy policy is optimal, but it is still worth comparing its performance with that of a simple heuristic. In the context of varying numbers of bursts, there is no reasonable fixed policy. The only way to guarantee stability would be to hire enough servers to cope with the maximum possible offered load, i.e. $n > J\lambda/\mu$, where J is the largest number of bursts allowed. That would clearly be too wasteful. However, one could envisage a simple dynamic policy that avoids searching for the best greedy decisions. If there are j bursts present, hire the minimum number of servers that can cope with the current offered load: $n = \lfloor j\lambda/\mu \rfloor + 1$. This will be referred to as the 'tight' policy.

Figure 8 shows the long-run average costs achieved by the tight and the greedy (optimal) policies, as the job arrival rate within bursts increases. Bursts arrive at rate $\gamma = 0.2$ and terminate at rate $\nu = 0.01$. Thus, there are about 20 bursts in progress on the average. The upper bound on the number

of bursts present is $J = 50$. The job service rate is $\mu = 5$ and λ varies from 10 to 30 jobs/sec. The holding and server costs are $c_1 = 0.5$ and $c_2 = 1$, respectively.

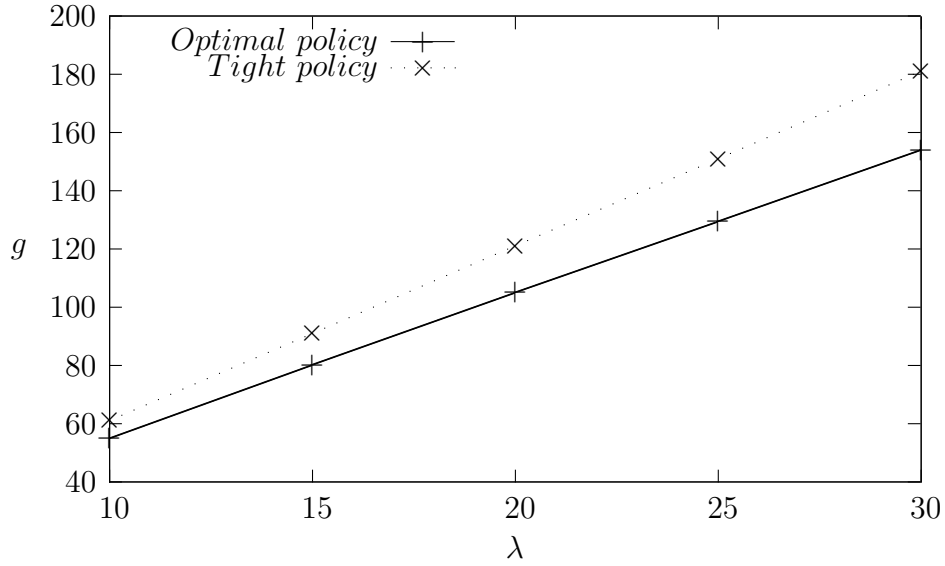

Figure 9: Bursty arrivals: Average cost against arrival rate within bursts

The figure shows that the costs achieved by the optimal policy are about 15% lower than those of the tight heuristic. Given that the computational complexity of determining the optimal policy is very small, that gain is worth having.

8. Conclusions

The problem of minimizing costs in a system where servers are hired dynamically was considered in the context of three traffic and hiring regimes: batch arrivals with arbitrary hiring intervals, Poisson arrivals with fixed hiring intervals and Poisson bursty arrivals. In all cases, the optimal hiring policy can be computed by applying a policy improvement algorithm. In addition, greedy heuristic policies are available which are often almost indistinguishable from the optimal policy. For the third regime the heuristic is the same as the optimal hiring policy.

One can envisage extending the models in several directions. For example, there may be jobs of different types, with different arrival and service characteristics and different holding and server costs. The system state at a decision epoch would then be a vector $(j_1, j_2, \ldots, j_k)$, where j_i is the number of jobs of type i present. The action taken at a decision epoch would also be a vector of server allocations, $(n_1, n_2, \ldots, n_k)$, where n_i is the number of servers hired to serve jobs of type i. The methodology described here would still apply, but the computation of the optimal policy would be considerably more complex. Another generalization would be to allow the traffic parameters λ and μ to change between decision intervals. They may depend on the current state, and possibly on the action taken, or may be controlled by a changing environment. Such systems could also be handled by the methods proposed here.

References

[1] M.N. Bennani and D. Menascé, "Resource allocation for autonomic data centers using analytic performance methods", Procs., 2nd IEEE Conf. on Autonomic Computing (ICAC-05), pp 229-240, 2005.

[2] P. Bodík, R. Griffith, C. Sutton, A. Fox, M. Jordan and D. Patterson, "Statistical machine learning makes automatic control practical for internet datacenters", Conf. on Hot Topics in Cloud Computing (HotCloud'09), Berkeley, CA, USA, 2009.

[3] P. Buchholz, J. Kriege and I. Felko, "Input Modeling with Phase-Type Distributions and Markov Models", SpringerBriefs in Mathematics, 2014.

[4] E.-K. Byun, Y.-S. Kee, J.-S. Kim, S. Maeng, "Cost optimized provisioning of elastic resources for application workflows", Future Generation Computer Systems, 27 (8) (2011) 1011 - 1026. doi:http://dx.doi.org/10.1016/j.future.2011.05.001.

[5] E.-K. Byun, Y.-S. Kee, J.-S. Kim, E. Deelman, S. Maeng, "BTS: Resource capacity estimate for time-targeted science workflows", Journal of Parallel and Distributed Computing, 71 (6) (2011) 848-862. doi:10.1016/j.jpdc.2011.01.008.

[6] A. Chandra, W. Gong and P. Shenoy, "Dynamic resourse allocation for shared data centers using online measurements", Procs., 11th ACM/IEEE Int. Workshop on Quality of Service (IWQoS), pp 381-400, 2003.

[7] S. Chaisiri, B.S. Lee and D. Niyato, "Optimization of resource provisioning cost in cloud computing", IEEE Transactions on Services Computing, 5(2), pp. 164–177, 2012.

[8] B.L. Fox and P.W. Glynn, "Computing Poisson Probabilities", Management Science and Operations Research, 31, 4, pp. 440-445, 1988.

[9] I. Gable, A. Agarwal, M. Anderson, P. Armstrong, K. Fransham, D. H. C. Leavett-Brown, M. Paterson, D. Penfold-Brown, R. J. Sobie, M. Vliet, A. Charbonneau, R. Impey, W. Podaima, "A batch system for HEP applications on a distributed IaaS cloud", Journal of Physics: Conference Series 331 (6) (2011) 062010.

[10] H. Hiden, S. Woodman, P. Watson and J. Cala, "Developing cloud applications using the e-science central platform", Royal Soc. of London, Phil. Trans. A. (Mathematical, Physical and Engineering Science), 371, 2013.

[11] H. Kim, S. Chaudhari, M. Parashar, C. Marty, Online risk analytics on the cloud, in: Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on, IEEE, 2009, pp. 484–489.

[12] U. Lampe, M. Siebenhaar, R. Hans, D. Schuller and R. Steinmetz, "Let the clouds compute: cost-efficient workload distribution in infrastructure clouds", in *Economics of Grids, Clouds, Systems, and Services*, Springer, 2012, pp. 91-101.

[13] W.E. Leland, M.S. Taqqu, W. Willinger and D.V. Wilson, "On the self-similar nature of Ethernet traffic", IEEE/ACM Transactions on Networking, 2, 1, pp. 1-15, 1994.

[14] M. Mazzucco, D. Dyachuk, and M. Dikaiakos, "Profit-aware server allocation for green internet services", IEEE Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 277–284, 2010.

[15] M. Mazzucco, I. Mitrani, M. Fisher and P. McKee, "Allocation and Admission Policies for Service Streams", Procs. MASCOTS'08, Baltimore, pp 155-162, 2008.

[16] M. Mazzucco, M. Vasar, and M. Dumas, "Squeezing out the cloud via profit-maximizing resource allocation policies", IEEE Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 19–28, 2012.

[17] A.S. McGough, M. Forshaw, C. Gerrard, S. Wheater, B. Allen and P. Robinson, "Comparison of a cost-effective virtual cloud cluster with an existing campus cluster", Future Generation Computer Systems, 45, pp. 65-78, 2014.

[18] I. Mitrani, "Managing Performance and Power Consumption in a Server Farm", Annals of Operations Research, DOI:10.1007/s10479-011-0932-1, 2011.

[19] I. Mitrani, *Probabilistic Modelling*, Cambridge University Press, 1998.

[20] A. Reibman and K. Trivedi, "Numerical transient analysis of Markov models", Computing and Operations Research, 15, 1, pp. 19-36, 1988.

[21] A. Rajabi and J.W. Wong, "MMPP Characterization of Web Application Traffic", 20th Int. Symp. on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'12), pp. 107-114, 2012.

[22] J. Revzina, "Possibilities of MMPP processes for bursty traffic analysis", Procs. 10th Int. Conf. on Reliability and Statistics in Transportation and Communication (RelStat10), Riga, pp. 131-135, 2010.

[23] D. Thain, T. Tannenbaum and Miron Livny, "Distributed computing in practice: the Condor experience", Concurrency and Computation: Practice and Experience, 17 (2-4),323-356, doi:http://dx.doi.org/10.1002/cpe.v17:2/4

[24] H.C. Tijms, *Stochastic Models*, John Wiley and sons, 1994.

[25] R. Urgaonkar, U. C. Kozat, K. Igarashi and M. J. Neely, "Dynamic Resource Allocation and Power Management in Virtualized Data Centers", IEEE/IFIP NOMS 2010, Osaka, Japan, 2010.

[26] K. Wang, M. Lin, F. Ciucu, A. Wierman and C. Lin, "Characterizing the impact of the workload on the value of dynamic resizing in data centers", Procs. 12th ACM SIGMETRICS/PERFORMANCE conf. on Measurement and Modeling of Computer Systems (SIGMETRICS'12), pp. 405-406, 2012.